Line-by-Line Assembler

COS'E' UN'ISTRUZIONE IN ASSEMBLER

da utilizzare sia con la MINIMEMORY
e sia con l' Editor ASSEMBLER.

FORMATO DELLE ISTRUZIONI IN LINGUAGGIO SORGENTE

Un programma sorgente in Assembler è formato da istruzioni in codice che possono contenere direttive di assemblaggio;istru= zioni di macchina,pseudo istruzioni commenti.

Ogni linea (o record) di istruzioni sorgente consiste in un massimo di 80 caratteri di informazioni inclusi gli spazi. Un record può essere suddiviso in più sezioni di lunghezza varia= bile chiamati campi.
Il CAMPO ETICHETTA (label) è posizionato all'inizio dell'istru= zione sorgente e serve come punto di riferimento.

Il CAMPO OP-CODE è il codice operativo(un numero, un nome, o un'abbreviazione) della azione che deve compiere l'istruzione sorgente.

IL CAMPO OPERANDO specifica il valore su cui l'istruzione agi= sce;può essere un numero, una stringa, un indirizzo etc etc.

IL CAMPO COMMENTO è un'area in cui potete aggiungere commenti per migliorare la leggibilità del programma ma esso non influ= enza le operazioni del computer.

Le definizioni di sintassi descrivono la forma richiesta per l'uso dei comandi in relazione ai campi.

LA SEZIONE 4 descrive le procedure di scrittura e definisce i dettagli.

Nelle definizioni sintattiche relative alle istruzioni di mac= china ed alle direttive di assemblaggio vengono usate le seguen= ti convenzioni. (vedi manuale del TI pag. 31) e

- Le parentesi angolari indicano un elemento da voi de= finito.
- Il simbolo b rappresenta uno o più spazi(o blanks)

La sintassi (cioè la forma richiesta) per l'istruzione sor= gente è la seguente:

[<label>] b op-code b [< operando >] [< operando>] b[<commento>]

Come indica questa definizione di sintassi, una istruzione sor= gente può avere una etichetta da voi definita. Uno o più spazi separano l'etichetta dall' op-code. Il generico termine op-code include codici operativi mnemonici, direttive di assemblaggio, e potete dunque inserire uno di questi elementi.

Uno o più spazi separano l' op-code dall'operando, quando è ri= chiesto un operando. Operandi addizzionali, quando servono, devo= no essere separati da virgole. Uno o più spazi separano l'opern= do o gli operandi dal campo commento. NOTA: Anche se la lunghez= za massima di un record sorgente è 80 caratteri, quando lo lista= te sono stampati solo i primi 60 caratteri di ogni linea.

CAMPO ETICHETTA

Il Campo Etichetta comincia col 1° carattere del record sorgente e termina al primo spazio seguente. Il campo etichetta è un sim= bolo contenente fino a sei caratteri, il primo dei quali DEVE es= sere alfabetico. Gli altri caratteri possono essere invece alfa= numerici. Una etichetta opzionale è per le istruzioni di macchina e per molte direttive di assemblaggio. Quando l'etichetta è omessa il primo carattere del record DEVE essere uno spazio.
Una istruzione SORGENTE formata solo da un CAMPO ETICHETTA è una istruzione valida. Ha l'effetto di assegnare la locazione corren= te all'etichetta. Di solito ciò è equivalente a piazzare l'etichet= ta nel campo etichetta della seguente istruzione di macchina o di= rettiva di assemblaggio.

CAMPO ISTRUZIONE

Il campo istruzione comincia dopo lo spazio che chiude il campo etichetta oppure nel primo carattere che non è uno spazio seguente la prima posizione del record se l'etichetta è omessa. Il campo istruzione termina con uno o più spazi e non può superare il 60° carattere del record sorgente.
Il campo istruzione contiene un op-code che può essere uno dei se= guenti:

°)codice operativo mnemonico di una istruzione di macchina.
°)codice operativo di una direttiva di assemblaggio
°)simbolo assegnato ad una "extended operation" da una direttiva DXOP
°)codice operativo di una pseudo istruzione.

CAMPO OPERANDO

Il campo operando comincia dopo lo spazio che conclude il campo is= truzione. Non può oltrepassare la 60° posizione del RECORD SORGENTE.
Il campo operando può contenere una o più espressioni,variabili,o costanti in accordo a ciò che richiede il particolare op-code. Il campo operando termina con uno o più spazi.

CAMPO o COMMENTO e linea di COMMENTO comincia dopo lo spazio che chiude il campo operando e può estendersi fino alla fine del re= cord sorgente se necessario. Il campo commento può contenere qua= lunque carattere ASCII incluso gli spazi . Unalinea o istruzione di commento consiste in un singolo campo che comincia con un asterisco e continua con qualunque carattere ASCII inclusi gli spazi, in qua= lunque ordine.
Le istruzioni di commento sono stampate nel listato del codice sor= gente ma non hanno altri effetti sull'assemblatore.Una linea com= posta esclusivamente di spazi è considerata una linea di commento.

A BREVISSIMA SCADENZA DAREMO TUTTE LE INDICAZIONI PER PROCEDERE SU QUESTO MERAVIGLIOSO TERRENO cioè IL LINGUAGGIO MACCHINA. IN PREPA= RAZIONE UN MANUALE IN ITALIANO CHE VERRA' CEDUTO ASSIEME ALLA SSS MINIMEMORY. LAVORIAMO IN 5 PER FARLO. PRONTO SARA' A ?NATA?F ?

Prima di inserire o staccare il modulo MINI MEMORY, é buona norma spegnere la consolle. Spegnendo la consolle si previene la possibilità della perdita o della modifica della memoria ad accesso diretto (RAM) del modulo.
NOTA; Assicuratevi che il modulo sia privo di elettricità statica prima di inse= rirlo nel computer (vedi "informazioni sul servizio e garanzia" per dettagli sull'elettricità statica).
I) Inserire il modulo nell'incastro sulla consolle. Poi accendete il computer e attendete che appaia sul video l'intestazione centrale.
2) Premete qualsiasi tasto per far apparire il menù di base. In questo menù compaiono 3 opzioni (I) TI BASIC (2) EASY BUG (3) MINI-MEMORY. Se scegliete EASY BUG, il programma EASY BUG di messa apunto(ricerca degli errori o modi= fica) viene caricato messo in funzionamento e compare sullo schermo il suo menù. Se selezionate MINI-MEMORY, potete scegliere fra 3 opzioni del suo menù (I) TO LOAD AND RUN (2) RUN (3) RE-INIZIALIZE.
LOAD AND RUN carica programmi assemblati in schede o disco nella memoria del modulo e lo esegue. RUN esegue i programmi precedentemente caricati in memoria, chiede il nome del programma da eseguire. RE-INIZIALIZE prepara il modulo a nuovi programmi o dati CANCELLA TUTTA LA MEMORIA DEL MODULO.
Quando scegliete questa opzione , lo schermo diventa momentaneamente vuoto poi appare il messaggio MEMORY ALREADY INIZIALIZED, HIT "PROC'D" TO CONFIRM premete "PROC'D" se volete confermare il comando. Reinizializzando si portano a O tutti i riferimenti a programmi esistenti e si prepara a caricare altri programmi o dati. NOTA Premete PROC'D soltanto se volete caricare un nuovo programma e lo spazio in memoria non è sufficente.

USO DEL MODULO MINI MEMORY COME ARCHIVIO FILE

Probabilmente le applicazioni più comuni del modulo consistono nella memoriz= zazione veloce e temporanea di dati, per uso di programmi TI-BASIC.
Dal momento che il modulo conserva i dati nella sua memoria (RAM) anche a consolle spenta può conservare una piccola quantità di dati 4K.
Potete se l'unitò di espansione di memoria è colelgata, aggiungere un nuovo tipo di file, EXPMEM2. Questo file può avere una lunghezza fino a 24Kbytes.
Il modulo MINI MEMORY introduce due nuovi files nel sistema
MINIMEM segmento di 4K RAM di memoria del modulo.
EXPMEM2 segmento di 24 K RAM situato nell'unità di espansione, disponibile solo se l'unità d'espansione è collegata.

ACCESSO AL FILE

Se volete questi files per la memorizzazione dati insieme a programmi in linguaggio ASSEMBLER dovete prendere certe precauzioni per evitare di distruggere dati o programmi, consultate MIXING per maggiori informazioni su questa procedura.
ATTENZIONE se i dati sono memorizzati nel modulo non si possono usare le possibilità del linguaggio ASSEMBLER.
-Struttura del file SEQUENTIAL o RELATIVE sequenziali o
-Modo di memorizzazione dei dati DISPLAY o INTERNAL
-Lunghezza del record VARIABLE o FIXED variabile o fissa.
-Modi di funzionamento INPUT OUTPUT UPDATE APPEND.
-Funzioni BASIC EOF (end of file) fine del file.

A queste descrizioni si devono applicare le seguenti limitazioni:
La lunghezza del record VARIABLE può essere usata solo con file SEQUENTIAL.
Per un file con records VARIABLE, un dato a lunghezza 0 nel primo record
sarà memorizzato in modo scorretto. Per garantire un appropriato funzionamento
assicuratevi che il primo record non sia una stringa nulla.PPer maggiori informa=
zioni sul modo di accedere ai file si rimanda a "TRATTAMENTO DEI FILES" sezi=
one guida per l'operatore.

FORMAZIONE DI UN FILE

Potete pensare ai files introdotti nel sistema a mezzo del modulo MINI MEMORY
come files molto veloci, fuori del programma di memorizzazione proprio come
una cassetta o un minidisco. I comandi da usare in TI BASIC sono gli stessi
di quelli descritti nella guida ad uso dell'operatore. Per accedere ad un file
dovete aprirlo con un comando OPEN elencando la descrizione del file che volete
aprire.
OPEN 3:("MINIMEM?RELATIVE?FIXED?UPDATE?DISPLAY)
I dati possono essere scritti nel file con una PRINT e possono essere letti
con un INPUT . L'istruzione RESTORE ricolloca il file al suo record iniziale.
Dovete chiudere il file quando non avete più bisogno di accedervi o se volete
riaprirlo per stabilire modi diversi (per esempio cambiarlo da OUTPUT a un
file di **tipo** INPUT.

FORMAZIONE DI FILES NELL'ESPANSIONE DI MEMORIA

Per formare un file nell'espansione di memoria si richiede la stessa proce=
dura descritta per il MODULO MINI MEMORY, con una eccezione. Per garantire
che un file sia aperto o chiuso in modo appropriato, ogni frase OPEN deve
essere preceduta da una istruzione CALL LOAD che specifichi un indirizzo
e un valore, (vedi subprogrammi addizionali TI BASIC per maggiori informazioni
sul subprogramma CALL LOAD) l'indirizzo è lo stesso per ambedue le istruzioni
CALL LOAD, il valore che segue l'indirizzo dipende dal tipo di file (INTERNAL
o DISPLAY) e dalla lunghezza del record (FIXED o VARIABLE).
Per files INTERNAL con records a lunghezza VARIABLE il formato è il seguente
CALL LOAD(-24574,24) seguito da OPEN n:("EXPMEM 2",SEQUENTIAL,VARIABLE32,
INTERNAL,OUTPUT
Per file tipo DISPLAY e records a lunghezza VARIABLE il formato è il seguente
CALL LOAD(-24574,I6) seguito da OPEN n:"EXPMEM 2",SEQUENTIAL,VARIABLE32,DISPLAY
OUTPUT
Per file tipo INTERNAL con records a lunghezza fissa il formato é il seguente
CALL LOAD(-24574,8) seguito da OPEN n:"EXPMEM 2",RELATIVE,FIXED,UPDATE,INTERNAL
Per file di tipo DISPLAY e records a lunghezza fissa il formato è il seguente
CALL LOAD(-24574,0) seguito da OPEN n:"EXPMEM 2",RELATIVE,FIXED,UPDATE,DISPLAY

LETTURA E SCRITTURA DI UN FILE

Iprogrammi seguenti spiegano la scrittura dei dati nei files MINIMEM e EXPMEM 2 e poi la lettura dei dati. Esempio MINIMEM:

```
I00 OPEN 5:"MINIMEM",SEQUENTIAL,FIXED,OUTPUT,INTERNAL
II0 INPUT X
I20 INPUT Y
I30 INPUT Z
I40 PRINT# 5:X,Y,Z
I50 CLOSE 5
```

Questo programma apre il file MINIMEM come un file di uscita nella linea I00. Le righe da II0 a I30 accettano i valori dei dati immessi dalla tastiera. La riga I40 registra questi valori nel file MINIMEM, e la riga I50 lo chiude.
A questo punto la consolle può essere spenta e il modulo può essere rimosso, i dati sono conservati proprio come se fossero stati memorizzati in cassetta o minidisco.
Il programma successivo legge i valori dei dati memorizzati nel file MINIMEM e li visualizza sullo schermo.

```
200 OPEN 5:"MINIMEM";SEQUENTIAL,FIXED,INPUT,INTERNAL
2I0 INPUT 5:P,Q,R
220 PRINT P,Q,R
230 CLOSE 5
```

ESEMPIO EXPMEM 2:

```
I00 CALL CLEAR
II0 REM OPEN FILE FOR DISPLAY-TYPE,VARIABLE-LENGTH
I20 CALL LOAD(-24574,I6)
I30 OPEN I:"EXPMEM2",SEQUENTIAL,VARIABLE,DISPLAY,UPDATE
I40 FOR I = I TO 20
I50 PRINT I:"RECORD ";I;"WAS READ."
I60 NEXT I
I70 RESTORE I
I80 FOR J =I TO 20
I90 INPUT I:AS
200 PRINT AS
2I0 NEXT J
220 CLOSE I
```

Questo programma apre un file in EXPMEM2, scrive 20 records nel file li legge di ritorno e li visualizza sullo schermo. Notate la frase CALL LOAD alla riga I20, che precede la frase OPEN alla Riga I30, e la frase RESTORE alla riga I70, che ricolloca il file I al suo records iniziale.

NOTA: quando la consolle è spenta, ogni dato memorizzato nell'unità di espan- sione memoria viene distrutto.

CARICAMENTO E SALVATAGGIO DI PROGRAMMI IN TI BASIC

Il modulo MINI MEMORY è anche utile per memorizzare brevi programmi in TI BASIC o in linguaggio ASSEMBLER. Questi ultimi memorizzati su minidischi sono caricati scegliendo LOAD AND RUN nella lista di selezzione del MINI MEMORY, mentre i programmi in TI BASIC possono essere salvati o caricati usando le istruzioni SAVE e OLD rispettivamente. P
Il modulo MINI MEMORY può memorizzare quasi 4K bytes (ESATTAMENTE 4088 bytes) di dati nella sua RAM.

CARICAMENTO E MEMORIZZAZIONE DI UN PROGRAMMA

La seguente procedura vi illustra come creare un programma -test- di una sola istruzione, conservarlo nel modulo, e poi ricaricarlo nella memoria della consolle. Per prima cosa, selezionate il TI BASIC ed immettete il programma.
IOO PRINT" THIS IS A TEST"
Memorizzate il programma immettendo l'instrizione SAVE MINIMEM
Dopo che il programma è stato memorizzato nel modulo potete spegnere la consolle. A questo punto, anche se si rimuove il modulo, il programma è conservato proprio come se fosse stato memorizzato su cassetta o mini- disco. Come controllo se non volete spegnere la consolle, immettete l'istruzione NEW per cancellare il programma. Per caricarlo nuovamente in memoria immettete l'istrùzione OLD MIMINEM. Per controllare che il programma sia stato ricaricato immettete l'istruzione LIST.

UNIONE DI PROGRAMMI ASSEMBLER E FILE TI BASIC

Programmi ASSEMBLER e file TI BASIC non possono essere memorizzati simultanea- mente nel modulo MINI MEMORY. Se il MODULO MINI MEMORY e l'unità di espansione di memoria sono entrambi disponibili, potete mescolare programmi in ASSEMBLER e programmi in TI BASIC con le seguenti,restrizioni:
Il modulo MINI MEMORY può essere usato solo per memorizzazioni in linguaggi ASSEMBLER, potete anche memorizzare programmi in ASSEMBLER nel segmento di 8K dell'unità di espansione.
Il segmento di 24K dell'unità di espansione di memoria può essere usato per file TI BASIC.
ATTENZIONE Se dati TI BASIC sono memorizzati nel modulo MINIMEM l'unità di espansione non può essere usata per programmi in ASSEMBLER. Se avete entram- bi sia il modulo MINIMEM che l'unità di espansione e volete mescolare program- mi in ASSEMBLER e in TI BASIC, usate il seguente procedimento per evitare la distruzione dei dati o dei programmi.
Inizializzate il modulo MINIMEM con EASY BUG e il comando RE-INITIALIZE, o con il TI BASIC con il comando INIT (vedi subprogrammi aggiuntivi)
Poi dal TI BASIC usate il comando OPEN per aprire un file nell'espansione di memoria (EXPMEM2) per la memorizzazione dei dati. Caricate i programmi in ASSEMBLER (vedi caricamento di programmi in ASSEMBLER).

SUBPROGRAMMI TI BASIC AGGIUNTIVI

Diversi subprogrammi inclusi nel modolo MINIMEM forniscono la possibilità di interfaccia tra programmi in linguaggio ASSEMBLER e in TI BASIC. Questi subprogrammi sono (INIT,LOAD,LINK,PEEK,PEEKV,POKEV,CHARPAT) ogni subprogramma è trattato in questa sezione.

SUBPROGRAMMA INIT
Formato : CALL INIT
La frase di chiamata del subprogramma INIT non ha argomenti oltre CALL INIT.
Raccomandiamo di usare generalmente CALL INIT nell'istruzione di programma o come comando prima dei subprogrammi LOAD o LINK. Il subprogramma INIT inizializza la memoria CPU per sottoprogrammi in linguaggio ASSEMBLER e reinizializza le tabelle interne nel modulo MINIMEM. Controlla se l'unità di espansione è collegata, se collegata imposta i corispondenti valori di tabel- la nel modulo MINIMEM per rendere possibile l'accesso all'unità di espansione.
ATTENZIONE:CALL INIT cancella tutti i programmi o dati dal modulo MINIMEM.
Dal momento che il modulo MINIMEM contiene una sua alimentazione interna il modulo non deve essere inizializzato ogni volta che la consolle centrale é accesa. Il subprogramma INIT deve essere usato solo se volete reinizializ- zare la memoria del modulo.
ATTENZIONE:Il modulo MINIMEM trattiene solo i dati contenuti nel modulo stesso ogni dato nell'unità di espansione va perduto al suo spegnimento.

SUBPROGRAMMA LOAD

Il subprogramma LOAD serve a due scopi. Carica programmi in ASSEMBLER nella memoria CPU. Carica direttamente dati nella memoria CPU.
Formato CALL LOAD("nome del file da caricare")
Questo formato carica programmi o dati in linguaggio ASSEMBLER per la successiva esequzione con il comando LINK. Il nome del file può essere ogni valida espressione di stringa e specifica quale file deve essere caricato.
Un codice in linguaggio macchina (programma prodotto da una conpilazione) rilocabile é caricato al primo indirizzo disponibile. Un objet code assoluto é caricato all'indirizzo assoluto specificato nell objet code (codice oggetto) Non viene riservato spazio a meno che la lunghezza non sia descritta nel campo "O-tag". Il caricamento di dati in memoria usata dall'interprete TI BASIC può causare il blocco del sistema. Se state usando solo il modulo MINIMEM senza l'unità di espansione collegata e accesa, il primo programma in linguaggio ASSEMBLER é caricato cominciando da >7II8 il minimo indirizzo disponibile nella RAM del modulo. Se l'unità di espansione é collegata e accesa il primo programma ASSEMBLER viene caricato a partire da >A000 l'indirizzo iniziale del più alto segmento di memoria dell'espansione. Successivi programmi sono caricati sequenzialmente.

Formato CALL LOAD(indirizzo,valore)
Quando il subprogramma LOAD viene usato per caricare dati nella memoria CPU dovrebbe essere specificata una lista di valori interi (chiamata lista POKE) La lista POKE dovrebbe iniziare con un indirizzo tra 0(>0000) e 32767(>7FFF) o un indirizzo tra -I(>FFFF) e -32768(>8000) seguita da una lista di numeri interi da usarsi come valore di un byte. Questa lista di dati viene caricata in locazioni consequtive a partire dall'indirizzo dato. Una stringa vuota("") separa l'ultimo byte di una lista POKE dalla successiva. L'indirizzo di una lista POKE é assoluto e i dati non sono rilocabili.
Se un programma é caricato tramite una lista POKE deve essere caricato anche il suo nome per essere poi chiamato con il comando LINK. Il nome del programma e il suo indirizzo sono aggiunti nella tavola REF/DEF nella memoria del modulo nel modo seguente. Primo, il primo indirizzo libero nel modulo (FFAM) e l'ultimo indirizzo libero nel modulo (LFAM) devono essere letti dalla memoria per mezzo del comando PEEK. Gli indirizzi di queste 2 variabili sono >70IC e >70IE rispettivamente. Dopo aver verificato che ci sia abbastanza spazio (8 byte) per aggiungere un'altra etichetta alla tabella REF/DEF, sottraete 8 dal vecchio LFAM e imettete il nuovo valore in LFAM >70IE usando l'istruzione CALL LOAD. Caricate il nome del programma (devono essere esattamente 6 bytes inclusi gli spazzi; poi l'indirizzo del programma (2 bytes) nello spazzio di 8 bytes aggiunti nella tavola REF/DEF.

SUBPROGRAMMA LINK

Formato CALL LINK("nome del programma")
Il subprogramma LINK trasferisce il controllo e,a scelta,una lista di para= metri da un programma TI BASIC a un programma ASSEMBLER. Il nome del program= ma é una espressione di stringa lunga da I a 6 caratteri che deve gia trovar= si nella tavola REF/DEF. La lista dei parametri é facoltativa. Questa lista é usata quando é necessario che i parametri siano passati tra il programma in linguaggio ASSEMBLER e il programma in TI BASIC. Potete far passare strin= ghe o variabili numeriche o espressioni.
Passaggio di parametri dal programma chiamante al programma chiamato.
In dipendenza del fatto se un parametro é una variabile o una espressione il parametro viene fatto passare con il nome o con il valore.
Se una variabile viene fatta passare ad un programma in linguaggio ASSEMBLER si può cambiare il suo valore nel linguaggio ASSEMBLER cosi pure cambiando il valore della variabile nel programma principale. Se la variabile in una lista di parametri non é apparsa in precedenti istruzioni TI BASIC, l'inter= prete crea una tavola dei simboli di entrata.
Le espressioni sono fatte passare con il valore, dal momento che non sono direttamente associate con una variabile. Il valore di una espressione non può essere fatto ripassare al programma chiamante. Quando un elemento di matrice come A(9), é dato nella lista dei parametri, viene fatto passare come una variabile. Una matrice completa può essere fatta passare facendo seguire il nome del parametro da parentesi. Se la matrice à più di una dimensione, si deve porre una virgola tra le parentesi per ogni dimensione aggiuntiva. Per esempio, A() indica una matrice numerica ad una dimensione. EXTS(,,) rappresenta una matrice stringa a 3 dimensioni. Per precisare che variabili certe si devono usare solo per far passare un valore, ma non per ritornare risultati, la variabile deve essere inclusa tra parentesi.
Per esempio, (SUMI) si riferisce al valore corrente della variabile numerica SUMI. (AS(5)) fa riferimento al valore dell'elemento matrice di stringa AS(5). Fate attenzione che matrici comlete non possono essere fatte passare con il valore ma devono essere fatte passare con il loro nome, cosi,(A()) non sarebbe ammissibile. Si può elencare in una lista di parametri un massimo di I5 elementi.
Il subprogramma LINK esegue le seguenti azioni:
Valuta il nome del programma in linguaggio ASSEMBLER e la sua lunghezza (da I a 6 caratteri) e colloca questa informazione nello stack di memoria. Costruisce la lista degli argomenti, consistente in identificatori per ogni argomento della lista dei parametri e costruisce un ingresso di memoria stack per ogni argomento. Trasferisce il nome del programma all'area dove il programma di servizio possa accedervi e trasferisce il controllo al pro= gramma di servizio. Di ritorno, salta ad una routine di errore, se un errore è stato riconosciuto. Altrimenti, riporta a 0 l'ingresso stack usato durante l'esecuzione LINK e ritorna al programma chiamante TI BASIC.

Il subprogramma LINK passa informazioni sugli argomenti via lista dei para= metri in CPU RAM e lo stack dei valori in VDP RAM.
Gli identificatori degli argomenti, identificatori di vecchi stack e il numero degli argomenti nella lista sono localizzati nelle seguenti posizi= oni CPU RAM: Indirizzo > 7002 -> 70II identificatore di argomento, un byte per ogni argomento. > 83I0 identificatore di un vecchio stack di valore dell'inte= rprete TI BASIC. > 83I2 numero degli argomenti nella lista dei parametri.
Il codice identificatore di argomento sono come segue: 0 espressione numeri= ca I espressione di stringa 2 variabile numerica 3 variabile di stringa 4 matrice numerica 5 matrice di stringa.
ESPRESSIONE NUMERICA La memoria di stack contiene il valore dell'espressione numerica. Il valore é espresso nollá registrazione in base I00. Il primo byte é l'esponente di I00. Se l'esponente é positivo é in eccesso di 64 Un esponente negativo é espresso con un valore minore di 64 nel primo byte. Gli altri 7 byte contengono da 0 a 99 in radice I00. Se il numero é negativo la prima voce (2 byte) é il conplemento dei due numeri.
ESPRESSIONE DI STRINGA Un ingresso di stringa in memoria stack consiste nelle seguenti informazioni: byte 0-I>00IC byte 2>65(l'identificatore di stringa usato dall'interprete TI BASIC) byte 4-5 l'indice al valore della stringa nella memoria VDP byte 6-7 la lunghezza della stringa (il byte 6 dovrebbe essere sempre 0 dal momento che la lunghezza massima di stringa é di 255 caratteri.
VARIABILE NUMERICA La memoria stack contiene le seguenti informazioni: byte 0-I l'indice all'ingresso della tabella dei simboli della variabile nella memoria VDP byte 2 zero byte 4-5 l'indicatore al valore di 8 byte della veriabile nella memoria VDP.
VARIABILE DI STRINGA Questo elemento é una variabile di stringa o un elemen= to di matrice di stringa. byte 0-I l'indicatore dell'ingresso della tavola dei simboli della variabile in VDP RAM byte 2 > 65(identificatore di stringa usato dall'interprete TI BASIC) byte 4-5 l'indicatore del valore di stringa nella memoria VDP byte 6-7 la lunghezza della stringa.
MATRICI NUMERICHE byte 0-I l'indicatore d'ingresso della tabella dei sim= boli della matrice nella VDP byte 2 zero byte 4-5 l'indicatore dello spazio della matrice numerica. Lo sapazio di valore per una matrice numerica ha 2 byte per ogni dimensione che indicano l'indice massimo per quella dimen= sione. I valori degli elementi sono memorizzati in ordine sequenziale.
MATRICE DI STRINGA Quest'ingresso é simile a ll'ingresso per le matrici numeriche, solo il byte contiene > 65. Lo sapazio di valore di una matrice di stringa contiene 2 byte per ogni dimensione indicante l'indice massimo seguito da un indicatore per ogni valore di elemento di matrice (valore di stringa) in VDP RAM. Notate che in una matrice numerica ogni elemento di matrice é memorizzato consecutivamente nella medesima area di memoria, mentre gli elementi di una matrice di stringa sono collocate in aree di memoria non continque.

NAME LINK ROUTINE (sottoprogramma chiamato da link)

Quando un sottoprogramma in ASSEMBLER é richiamato dal TI BASIC per mezzo di una istruzione CALL LINK ,il controllo passa ad un sottoprogramma attra= verso una routine NAME LINK posta in un programma di servizio. Il programma chiamato da LINK trova il nome del programma nella tabella REF/DEF posta nell'ultima parte della memoria del modulo. Quando si carica un programma in ASSEMBLER il programma di caricamento aggiunge un ingresso di 8 byte alla tabella REF/DEF quando vede un'etichetta di REF o DEF. Questa tabella REF/DEF comincia da > 7FFF e scende > 7II8, il primo indirizzo libero (FFAM) nel modulo. La tabella REF/DEF é ricercata dall'indirizzo minimo. Perciò se si caricano 2 programmi col medesimo nome, si usa il secondo. Se il nome che fornite é più grande di 6 caratteri oppure se il programma chiamato da LINK non può trovare il nome nella tabella, ne risulata un errore. Il programma chiamato da LINK trasferisce il controllo al programma in ASSEMBLER con un comando 9900 branch-and-link (BL). Quando, da un sottoprogramma LINK, si chiama un programma in ASSEMBLER, l'area di lavoro é posta a > 70B8 e l'indirizzo di ritorno é in RII di quell'area di lavoro. Prima di ritor= nare, il vostro programma dovrebbe cancellare il byte a > 837C; altrimenti anche se il programma non à prodotto errore può essere visualizata una segnalazione di errore. Il programma in ASSEMBLER può attribuire nuovi valori a variabili numeriche o di stringa oppure ad elementi di matrici numeriche o di stringa con programmi di utilità forniti dal sistema. Questi programmi di utilità sono descritti nella sezione" routine di gestione sistema ". Ingressi sullo stack di valore che risultano da parametri fatti passare a mezzo dell'istrúzione CALL LINK, sono cancellati automaticamente dal sub= programma LINK. Se manipolate direttamente lo stack di valore, comunque, dovete ripristinare lo stack al suo stato originale prima di ritornare il controllo al subprogramma LINK.

SUBPROGRAMMA PEEK

Formato: CALL PEEK(indirizzo,variabile)

Il subprogramma PEEK é usato per leggere i byte di CPU RAM direttamente nelle variabili TI BASIC. Il parametro di indirizzo deve essere o un'espres= sione numerica o una variabile numerica. l'indirizzo é un valore decimale da -32768 a 32767, che rappresenta un valore intero di due byte. Indirizzi sopra > 7FFF sono scritti come numeri negativi, trattando il valore come un numero intero complemento dei 2. (Per esempio, per accedere a un indirizzo sopra 32767, sottraete 65536). La lista delle variabili deve consistere di variabili numeriche. Ogni byte conseqotivo letto dalla memoria é assegnato ad ogni variabile nell'ordine elencato nella lista delle variabili. Una stringa nulla ("") separa u a sequenza PEEK dalla successiva cosicchè potete leggere ripetutamente diverse posizioni di memoria con una singola istruzione. Per esempio, l'istruzione CALL PEEK(8I92,A,B,C(8),"",24576,X) legge tre bytes dall'indirizzo > 2000 e su; assegna i valori alle variabili A,B,C(8) consecutivamente; legge un byte dalla posizione > A000; e memorizza il valore nella variabile X. Il valore restituito é un valore di un byte ed é sempre conpreso tra 0 é 255.

SOTTOPROGRAMMA PEEKV

Formato CALL PEEKV(indirizzo,variabile)
Il sottoprogramma PEEKV é usto per leggere bytes da VDP RAM. Lavora esatta=
mente come PEEK, tranne che PEEKV accede a VDP RAM invece che CPU RAM.
L'indirizzo é un valore decimale da 0 a I6383, e la lista delle variabili
é una lista di variabili numeriche che vengono a contenere i valori letti.
Notate che il VDP ha I6K di RAM, e tentando di accedere ad un'indirizzo
di memoria più alto di I6383 potreste dannegiare il sistema. Vedere il
sottoprogramma PEEK per maggiori inforamazioni.

SOTTOPROGRAMMA POKEV

Formato CALL POKEV(indirizzo;variabile)
Il sottoprogramma POKEV vi permette di modificare il valore nel VDP RAM.
lavora nello stesso modo di LOAD quando si usa LOAD si modifica la CPU RAM.
L'indirizzo é un valore decimale fra 0 a I6383, e var é una espressione
numerica o una variabile numerica che contiene un valore da porsi nella
memoria VDP all'indirizzo specificato. Ogni valore specificato é memorizzato
consecutivamente a partire dall'indirizzo dato. Per esempio, CALL POKEV(784,
30,30,30,"",2,V) cambia la tavola dei colori I6 nella tabella dei colori I8
(indirizzo>3I0 a>3I2 in VDP RAM), risolvendosi in un primo piano nero
e sfondo grigio. Se il valore di V é I62, il carattere "A" appare nell'ango=
lo in alto a sinistra dello schermo.

SUBPROGRAMMA CHARPAT

Formato CALL CHARPAT(char-code,str-var)
Il subprogramma CHARPAT rimanda un identificatore di carattere (pattern)
di I6 caratteri che specifica la configurazione del codice di carattere.
Il codice di carattere é ogni numero di carattere compreso tra 32 e I59.
I codici di carattere da 32 a 95 (fino a I27 sul TI 99/4A) sono normalmente
riservati ai caratteri ASCII e definiti inizialmente dall'interprete TI BASIC.
L'espressione di stringa del codice di carattere é letta nella variabile di
stringa (str-var). Questa espressione consiste di I6 caratteri di cifre
esadecimali che rappresentano il carattere.

CARICAMENTO DI PROGRAMMI ASSEMBLER

Se usate solo il modulo MINIMEM senza l'unità di espansione, il vostro
programma carica direttamente nella RAM del modulo. Il primo programma
del linguaggi ASSEMBLER é caricato apartire da>7I18 che é l'indirizzo
disponibile più basso nella RAM del modulo. Alle volte si può desiderare
di caricare un programma direttamente nel modulo mentre l'unità di espansione
é collegata, oltrepassando la normale sequenza di caricamento. Per farlo
é necessario rendere l'unità di espansione temporaneamente "invisibile"
al sistema cancellando i valori nelle posizioni >7022 fino a >7029.
Questi sono i valori che indicano la presenza dell'unità di espansione.
Il modo più facile consiste nell'usare 2 comandi LOAD, uno con una lista
POKE e uno che carichi il programma in ASSEMBLER nel modulo.

PROGRAMMI DI SERVIZIO DEL SISTEMA

I programmi di servizio situati nel modulo MINIMEM possono essere richiamati da un programma in ASSEMBLER per accedere alle risorse dell'elaboratore e interfacciare con l'interprete TI BASIC. Nel modulo MINIMEM sono forniti due tipi di programmi di servizio . Un programma contiene una raccolta di programmi di utilità con cui collegarsi a routines ROM/GROM, esegue un esame della tastiera, accede al VDP ecc.
Un secondo programma contiene programmi di utilità di interfaccia in TI BASIC con cui un programma in ASSEMBLER può accedere a variabili passate attraverso un'istruzione CALL LINK . Questo programma contiene anche una routine di ricerca errore per rimandare eccezioni al TI BASIC.

PROGRAMMI STANDARD DI UTILITA
Tutti i programmi di servizio usano UTILWS (indirizzo > 7092)
per registri di zona lavoro e tutti i parametri sono passati attraverso i registri della zona lavoro del programma chiamante.Per vostra comodità USRWSP (indirizzo > 7088) é riservato per l'insieme dei registri di zona lavoro del programma. Comunque ogni area di registro che fornite può essere usata per passare parametri.
Le seguenti sezzioni descrivono le convenzioni sul passaggio dati e la sintassi della frase di richiamo per ogni routine.
VDP single byte write-VSBW
Formato BLWP @VSBW uguaglia VSBW a >6024
Questo programma scrive un valore di un singolo byte su uno specifico indirizzo VDP RAM. RO L'indirizzo VDP RAM. RI Un valore di un byte nel byte più significativo del registro I.
VDP Multiple byte Write-VMBW
Formato: BLWP @VMBW uguaglia VMBW a>6028.
Questa routine scrive bytes multipli da CPU RAM a VDP RAM. RO indirizzo VDP RAM.RIindirizzo di partenza del BUFFER CPU RAM.R2 numero di bytes da scrivere.
VDP Single byte read-VSBR
Formato: BLWP @VSBR uguaglia VSBR a > 6C2C
Questa routine legge un singolo byte da un'indirizzo VDP RAM specificato.
RO indirizzo VDP RAM. RI il valore letto da VDP RAM nel byte più significativo.
VDP Multiple bytes read-VMBR
Formato BLWP @VMBR uguaglia VMBR > 6C30
Questa routine legge bytes multipli da VDP RAM in CPU RAM.
RO indirizzo VDP RAM da cui leggere. RI indirizzo di partenza del BUFFER CPU RAM. R2 numero di bytes da leggere.
VDP Write to register- VWTR
Formato: BLWP OVWTR uguaglia VWTR a > 6C34.
Questa routine scrive un valore di un singolo byte in ogni registro del VDP RAM. RO il byte meno significativo contiene il valore che deve essere scritto; il byte più significativo contiene il numero di registro VDP (da 0 a7) dove scrivere.

KEYBOARD SCAN - KSCAN

Formato: BLWP @KSCAN uguaglia KSCAN >6020
Questo programma esamina una tastiera specificata e ritorna un codice chiave e uno stato. Le posizioni di memoria seguenti sono usate per comunicazioni tra il programma utente e la routine. >8374 numero dell'unita di tastiera . Questo numero di un byte deve essere specificato dal vostro programma . Il significato di questo byte é lo stesso dell'unità chiave nel sotto programma TI BASIC KEY. >8375 ASCII valore del tasto premuto (un byte). >8376 telecontrollo posizione Y (un byte). > 8377 telecontrollo X (un byte). >837C GPL registro di stato (un byte). Il byte di stato GPL può essere provato prima che il codice chiave sia letto. Potete farlo con una istruzione COC . Il bit 5 del byte di stato é settato se é stato pramuto un tasto nell'ultima chiamata KSCAN. I bytes di stato GPL sono assegnati come segue:

H	GT	COND	CARRY	OVF	0	0	0
7	6	5	4	3	2	I	0

PROGRAMMI DI UTILITA ESTESA

Programmi di utilità estesi sono forniti per accedere a routines nella consolle GROMs e ROMs. Questi programmi sono GPLLNK (collegamento a routines di GPL in GROM), XMLLNK (collegamento a routines in ROM), e DSRLNK (collegamento a routines delle unità periferiche). Usando questi programmi dovete assicurarvi che i registri di lavoro GPL non siano cambiati, che lo spazio di memoria usato dalla routine della consolle sia impostato appropriatamente, che la routine ritorni correttamente al programma.

Collegamento a routines GROM situate in memoria. - GPLLNK

Formato: BLWP @GPLLNK uguaglia GPLLNK a >60I8
DATA consolle routine address
La routine GPLLNK imposta un flag interno per indicare che un programma GPL é stato chiamato da un programma in linguaggio ASSEMBLER, carica l'area di lavoro GPL(indirizzo >83E0), salta al codice GROM ed esegue la routine GPL specificata da DATA. La routine GPL deve ritornare con un comando RTN per ritrasferire il programma al programma chiamante. Quando nella routine GPL si trova comando RTN, questo ritorna al programma del sistema. Il programma del sistema controlla il flag interno e trovandolo settato ritorna alla routine del linguaggio ASSEMBLER. Alcuni degli indirizzi delle routines GPL e le loro convenzioni di chiamata e di ritorno sono dati sotto. I nomi FAC, STACK, e STATUS sono usati nelle seguenti descrizioni. FAC é assegnato a >834A, STACK a >836E, STATUS a >837C. STATUS é il byte di stato GPL. E'organizzato nel modo seguente:

	HIGH	GREATER	CONDITION	CARRY	OVERFLOW	UNUSED
Bit	7	6	5	4	3	2,I,0

Bit 7 bit alto. Controllato durante l'esecuzione dell'interprete GPL.
Bit 6 Controllato dall'interprete GPL durante l'esecuzione del programma GPL.

Bit 5 Bit di stato. Controllato dall'interprete GPL. La routine keyscan inserisce questo bit quando si trova una nuova chiave. Anche la routine DSR inserisce questo bit per indicare che non esiste un file.
Bit 4 Bit di riporto. Controllato dall'interprete GPL.
Bit 3 Bit di OVERFLOW. """ " " " " " " " " " " " " .

La direttiva DATA specifica l'indirizzo della routine GPL da eseguirsi. Ogni routine é descritta sotto.

DATA >0016 Carica l'insieme di caratteri standard in VDP RAM.
Input: FAC - indicatore dell'indirizzo di partenza in VDP RAM dove vengono caricati i caratteri.
Output: VDP RAM all'indirizzo specificato in FAC contiene l'insieme di caratteri standard.

DATA >0018 Carica l'insieme di caratteri piccoli (per il modo TEXT) in VDP RAM
Input: Lo stesso di DATA 0016.
Output: " " " " " " " " " " " " .

DATA >0020 Inserisce ed inizializza il sistema.
Input: nessuno
Output: il sistema é inserito e inizializzato. Il suono e i circuiti VDP sono cancellati; i valori di DEFAULT per i registri VDP, l'insieme di caratteri, la tabella dei colori, e il blocco di stato sono caricati. L'ampiezza VDP RAM disponibile é memorizzata in>8370.

DATA >0034 Tono di accettazione. Emette un tono di accettazione per input. Non si richiede nessuna predisposizione di memoria prima di chiamare la routine.

DATA >0036 Tono di risposta errata. Emette un tono di risposta errata. Non si richiede predisposizione di memoria per la chiamata di routine.

DATA >0038 Get string space routine. Assegna uno spazio di memoria in VDP RAM con un numero di bytes specificato. Questa routine non dovrebbe essere usata al di fuori delle operazioni TI BASIC.
Se non c'é abbastanza spazio, la routine fà una "raccolta di rifiuti" per eliminare temporaneamente stringhe e poi riprova. Se non c'é ancora abbastanza spazio la routine emette un messaggio di errore MEMORY FULL.
Input: Gli indirizzi >830C e >830D dovrebbero contenere il numero di bytes che devono essere assegnati.
Output: L'indirizzo >831C indica lo spazio di stringa assegnato e l'indirizzo >831A indica il primo indirizzo libero in VDP RAM.
I 4 bytes agli indirizzi da>8356 a >8359 sono usati da questa routine. L'area FAC può essere distrutta se si fa una "raccolta di rifiuti."
Nota: Benche questa routine sia destinata ad assegnare uno spazio di stringa in VDP RAM, é anche utile per assegnare spazio per il PAB e per data BUFFER richiesti da un DSR.

DATA > 003B Routine di bit inverso. Procura un immagine speculare di un byte di informazione. E'usata molto comunemente per formare un'immagine speculare di una definizione di carattere.

Input: FAC- indirizzo dei dati in VDP RAM.

FAC + 2 - numero di bytes da ivertire.

Output: Il numero specificato di bytes in VDP RAM sono bit invertiti; cioé bits 0 e 7, I e 6, 2 e 5, 3 e 4 sono scambiati.

Effetti secondari: CPU RAM da> 8300 a >8340 é cancellato.

DATA >003D Routine di sevizio di unità in cassetta. Accede alla routine DSR di cassetta.

Input: Il PAB e il data BUFFER devono essre predisposti in VDP RAM prima della chiamata. L'offset dello schermo é >60 per il TI BASIC e >00 al di fuori della configurazione TI BASIC. L'indirizzo di partenza dello schermo deve essere >00 per le risposte emesse dalla cassetta DSR. FAC é il nome dell'unità (per es. CSI). L'indirizzo >8356 indica il primo carattere dopo il nome nel PAB. Gli indirizzi >8354 e >8355 sono la lunghezza del nome (per es.>0003 per CSI). La voce all'indirizzo >8300 dovrebbe essere posta a>0000. L'indirizzo >8360 deve essere posto a>08 ad indicare una chiamata DSR. Il byte di stato deve essere>00.

Cutput: Le risposte DSR per l'operazione della cassetta.

DATA >004A Carica l'insieme di minuscole in VDP RAM.

Input e output sono le stesse che per il caricamento di altri set di caratteri. NOTA: questa routine si applica solo alla consolle TI - 99/4A.

Uno degli usi per la routine di collegamento GPL é chiamare i programmi in virgola mobile (floating - point routines)scritti in GPL da un programma ASSEMBLER. Il contenuto delle posizioni CPU RAM da >834A a >836F può essere usato quando questi programmi in virgola mobile sono chiamati, e le posizioni VDP RAM da >03C0 a >03DF sono usate come area di BUFFER. Il byte di stato GPL riflette la condizione di caloolo. Tutti i valori dati di input output sono in formato virgola mobile. Quando capitano errori durante l'esecuzione di programmi in virgola mobile, sono indicati nella posizione CPU RAM >8354. I codici di errorre sono dati sotto.

CI errore di eccedenza, 02 errore di sintassi, 03 eccedenza intera in conversione, 04 radice quadrata di un numero negativo, 05 numero negativo elevato a potenza non intera, 06 logaritmo di un numero negativo o 0, 07 argomento non ammissibile in funzione trigonometrica. I programmi in virgola mobile sono descritti sotto.

DATA > 00I4 Converte un numero a virgola mobile in una stringa ASCII

Input: FAC - valore a virgola mobile di 8 byte.

FAC + II - se posizionata a 0, la stringa di uscita é in formato BASIC altrimenti l' uscita é in modo fix che richiede dati in FAC + I2 e FAC + I3. FAC + I2 se I, esprime eccedenza dal campo di calcolo con +- EE...E.. Il supero negativo di capacita é espresso come 0.

FAC + I3 il numero di cifre da fissare alla destra del punto decimale Un valore negativo disattiva il modo operativo fix.

Output: FAC - modificato

FAC + II il byte meno significativo dell'indirizzo dove é posizionata la stringa risultante. Il valore > 8300 deve essere aggiunto per otte-

nere l'indirizzo reale. FAC + I2 la lunghezza della stringa in bytes.

DATA >0022 la più grande funzione intera (INT) - calcola il più grande numero intero contenuto nel valore.
Input: FAC il valore a virgola mobile.
Output: Il risultato.
Per numeri positivi, il numero intero é il valore arrotondato per difetto. Per numeri negativi, l'intero é il valore arrotondato per difetto +I. STATUS insieme in accordo con il risultato.

DATA >0024 Routine di elavozione a potenza (PWR). Eleva un numero ad una potenza specificata.
Input: FAC il valore dell'esponente. STACK l'indicatore di STACK in VDP RAM che contiene il valore di otto byte.
Output: FAC il risultato in formato virgola mobile.
Questo é calcolato come EXP (valore dell'esponente) LOG(ABS(base)).
STATUS insieme in accordo con il risultato. Condizioni di errore: numero negativo elevato a potenza non intera, e 0 elevato a potenza negativa.
Effetti secondari: le posizioni >8375 e >8376 sono distrutte e il contenuto di una parola di >836E é diminuita di 8. Anche gli indirizzi FAC + I2 fino a FAC + I9 sono distrutti.

DATA >0026 Routine di radice quadrata (SQR) calcola ra radice quadrata di un numero.
Input: FAC il valore di ingresso.
Output: FAC la radice quadrata del valore di ingresso.
STATUS insieme in accordo al risultato.
Effetti secondari: gli indirizzi >8375 e >8376 sono distrutti.

DATA >0028 Routine dell'esponente (EXP) calcola il LOG naturale inverso del valore di ingresso.
Input: FAC il valore di ingresso.
Output: FAC il valore risultante.
STATUS insieme in accordo con il risultato.
Effetti secondari gli indirizzi >8375 e >8376 sono distrutti

DATA >002A Routine di LOG naturale (LOG) calcola il LOG naturale di un numero.
Input: FAC valore di ingresso.
Output: FAC il LOG naturale del valore di ingresso.
STATUS insieme in accordo con il risultato.
Effetti secondari gli indirizzi>8375 e>8376 sono distrutti.

DATA >002C Routine del coseno (COS) calcola il coseno di un numero
Input: FAC il valore di ingresso.
Output: FAC il coseno del valore di ingresso.
STATUS insieme in accordo con il risultato.
Effetti secondari gli indirizzi >8375 e >8376 sono distrutti.

DATA >002E Routine del seno (SIN) calcola il seno di un numero
Input: FAC valore di ingresso.
Output: FAC il seno del valore di ingresso.
STATUS in accordo con il risultato.
Effetti secondari gli indirizzi >8375 e >8376 sono distrutti.

DATA > 0030 Routine di tangente (TAN) calcola la tangente di un numero.
Input: FAC il valore di ingresso.
Output FAC la tangente del valore di ingresso.
STATUS in accordo con il risultato.
Effetti secondari gli indirizzi >8375 e >8376 sono distrutti.

DATA > 0032 Routine di arco tangente (ATN) calcola l'arco tangente di un numero.
Input FAC la grandezza di ingresso.
Output FAC l'arco tangente del valore di ingresso.
STATUS in accordo con il risultato.
Effetti secondari gli indirizzi >8375 e >8376 sono distrutti.

COLLEGAMENTO A ROUTINES RESIDENTI IN ROM - XMLLNK

Formato: BLWP @XMLLNK assegna XMLLNK a >60IC
DATA codice di routine della consolle.

Si può accedere alle routines nella consolle ROM attraverso la routine XMLLNK. Potete accedere ad una routine in consolle ROM in due modi.
Un modo consiste nello specificare il codice di routine in una istruzione DATA . Il byte basso dell'istruzione DATA dovrebbe essere posto a 0.
Nella tabella seguente viene data una lista dei codici di routine XML che possono essre chiamati da un programma ASSEMBLER.

Codice	Nome	Funzione
06	FADD	Addizione in virgola mobile.
07	FSUB	Sottrazione " " ".
08	FMULT	Moltiplicazione " ".
09	FDIV	Divisione " " ".
0A	FCOMP	Operazione di confronto in virgola mobile.
0B	SADD	Addizione di stack " " " " " .
0C	SSUB	Sottrazione di stack " " " " .
0D	SMULT	Moltiplicazione di stack " " " " .
0E	SDIV	Divisione " " " " " "" .
0F	SCOMP	Confronto " " " " " " .
I0	CSN	Converte una stringa in numero.
I2	CFI	Converte il formato a vircola mobile in un intero.
I7	VPUSHG	Colloca un valore nello stack di valore.
I8	VPOP	Preleva un valore dallo stack di valore.
23	CIF	Converte un intero in virgola mobile.

Il codice XML, che é un byte singolo é diviso in un alto semi byte che contiene l'indirizzo di tabella XML, e in un basso semi byte che contiene l'indice dentro quella tabella. Ci sono I6 indirizzi di tabella definiti nello spazio di indirizzi CPU. L'alto semi byte specifica da quale delle I6 tabelle prendere l'indirizzo di arrivo, e il basso semi byte determina quale dei I6 indirizzi nella tabella deve essere usato. Ogni tabella può contenere fino a I6 indirizzi di punto di ingresso di 2 byte.

Un altro modo di accedere ad una routine nella consolle ROM é di specificare il suo indirizzo nell'istruzione DATA. Notate che il bit alto della parola DATA deve essere posto cosi che il programma del sistema riconosca questo dato come un indirizzo e non come un codice XML. Per esempio BLWP aXMLLNK DATA >8D3A salta all'indirizzo ROM di consolle >0D3A che é un programma di confronto in virgola mobile. FAC (l'accumulatore in virgola mobile) prende avvio all'indirizzo >834A; ARG (che contiene argomenti) prende avvio all'indirizzo >835C, e STACK é all'indirizzo >836E. Il byte di STATUS é all'indirizzo >837C. Ogni errore di eccedenza, eccetto in CFI, ritorna 0I all'indirizzo >8354.

DATA >0600 Addizione in virgola mobile (FADD) - addiziona due valori.
Input: FAC primo valore. ARG secondo valore. Output. FAC risultato.

DATA >0700 Sottrazione in virgola mobile (FSUB) sottrae due valori.
Input: FAC valore che deve essere sottratto. ARG valore da cui FAC é sottratto. Output: risultato.

DATA >0800 Moltiplicazione in virgola mobile(FMULT) moltiplica due valori.
Input: FAC moltiplicatore. ARG moltiplicando. Output: risultato.

DATA >0900 Divisione in virgola mobile (FDIV) divide due valori.
Input: FAC divisore. ARG dividendo. Output: FAC risultato.

DATA >0A00 Conparazione in virgola mobile (COM) confronta 2 numeri in virgola mobile. Input: ARG primo argomento. FAC secondo argomento.
Output: STATUS insieme in accordo con il risultato. Il bit alto é settato se ARG é logicamente più alto di FAC.
Il più grande di un bit é settato se ARG é arittmeticamente più grande di FAC. Il bit= é settato se ARG e FAC sono uguali.

DATA >0B00 Addizione di STACK di valore (SADD)addizionausando uno STACK In VDP RAM. Input: STACK - indirizzo in VDP RAM dove é locato il termine di partenza. FAC valore terminale. Output: FAC risultato.

DATA> 0C00 Sottrazione di uno STACK di valore (SSUB) sottrae usando uno STACK in VDP RAM. Input STACK l'indirizzo VDP RAM che contiene il termine di partenza. FAC valore che deve essere sottratto. Output FAC risultato della sottrazione.

DATA >0D00 Moltiplicazione di STACK di valore (SMULT) moltiplica usando uno STACK in VDP RAM. Input: STACK indirizzo VDP RAM che contiene il moltiplicando. FAC valore moltiplicatore. Output risultato.

DATA >0E00 Divisione di STACK di valore (SDIV) divide usando uno STACK in VDP RAM. Input: STACK indirizzo in VDP che contiene il dividendo. FAC valore divisore. Output: FAC: risultato.

DATA >0F00 Confronto di STACK di valore (SCOMP) confronta un valore nello STACK VDP RAM con il valore in FAC. Input: STACK indirizzo VDP che contiene uno dei valori da comparare. FAC altro valore da comparare.
Output: STATUS insieme in accordo con il risultato.Il bit alto é settato se il valore indicato dallo STACK é logicamente più alto di FAC. Il più grande di un bit é settato se il valore indicato da STACK é aritmeticamente più grande di FAC. Il bit= é settato se i valori indicati da STACK e FAC sono uguali.

DATA >1000 Converte stringhe in numeri (CSN) - converte una stringa ASCII in un numero a virgola mobile. Input: FAC + I2 - indirizzo della stringa in VDP RAM. Output: FAC - risultato.

DATA > I200 Converte un numero a virgola mobile in un intero (CFI). Input: FAC - numero a virgola mobile da convertire. Output: FAC - il valore intero . Il valore massimo é>FFFF. Se capita un overflow FAC + IO (>8354) é settato in un codice di errore> 03 .

DATA > I700 Colloca il valore nello STACK di valore (VPUSHG) Colloca un valore da FAC nello STACK.

DATA > I800 Preleva un valore dallo STACK di valore e lo pone in FAC (VPOP).

DATA> 2300 Converte un intero in un numero a virgola mobile (CIF) Input: FAC il valore intero di una parola da covertire. Output: FAC risultato.

COLLEGAMENTO A PROGRAMMI DI SERVIZIO DELL'UNITA - DSRLNK

Formato: BLWP @DSRLNK uguaglia DSRLNK a > 6038
DATA codice di routine di consolle.

DSRLNK aggancia un programma in linguaggio ASSEMBLER ad alcuni programmi di servizio dell'unità (DSR) o a sotto programmi in ROM. I dati forniti sono 8 per il collegamento ad un programma di servizio DSR e IO per il collegamento ad un sottoprogramma. Prima di chiamare questa routine, si deve predisporre un PAB in VDP RAM. Un PAB é un blocco di memoria che contiene informazioni sul file cui si deve accedere. In aggiunta, gli indirizzi CPU RAM da >8356 a>8357 devono contenere un indicatore della lunghezza del nome dell'unità periferica o del sottoprogramma nel PAB. Dopo che la routine é stata eseguita l'informazione é ripassata al vostro programma in linguaggio ASSEMBLER nell'area UTLTAB. NOTA : dal momento che la cassetta BSR é nell GROM, vi si deve accedere attraverso GPLLNK piuttosto che DSRLNK. Per accedere add una cassetta, usate BLWP @GPLLNK con DATA > 003D.

PROGRAMMI DI UTILITA DI INTERFACCIA TI BASIC

Questi programmi permettono ad un programma il ASSEMBLER di leggere o assegnare valori alle variabili passate in una lista di parametri da una istruzione CALL LINK in un programma TI BASIC. Queste routine di gestione sistema includono programmi di utilità di passaggio di argomenti e un programma di segnalazione di errori. Ogni programma di passaggio di argomenti usa il suo proprio spazio di lavoro, localizzato a >7092. Comunque, ogni parametro é passato attraverso lo spazio di lavoro del programma chiamante.
Le seguenti sezioni descrivono le convenzioni di passaggio dati e la sintassi della istruzione di chiamata per ogni routine.

ASSEGNAZIONE NUMERICA NUMASG

Formato: BLWP @NUMASG uguaglia NUMASG a >6040

Questa routine assegna un valore numerico ad una variabile numerica passata come un argomento. RO zero se si usa una variabile numerica semplice oppure un numero di elemento di matrice se si fa un assegnamento ad un elemento di matrice. RI numero di argomento come appare nella lista di CALL LINK.
834A area FAC contiene un valore a virgola mobile di otto byte da assegnare alla variabile. Se l'argomento richiesto non é una variabile numerica o un elemento di matrice numerica, viene emessa una segnalazione di errore.

ASSEGNAZIONE DI STRINGA - STRASG

Formato: BLWP @STRASG uguaglia STRASG a >6048

Questa routine assegna una stringa a una variabile di stringa passata come un argomento al programma in linguaggio ASSEMBLER. Questa routine: riserva spazio per la stringa in VDP RAM. Copia la stringa nel VDP RAM riservato. Assegne la stringa alla variabile selezionata. Modifica l'ingresso di STACK dell'argomento originale per indicare la nuova stringa. La stringa che deve essere assegnata, deve essere creata in RAM dal programma in linguaggio ASSEMBLER. Il primo byte della stringa contiene la lunghezza della stringa. Ai registri vengono atribuiti i seguenti valori. R0 zero se una stringa é assegnata a una variabile di stringa semplice oppure un numero di elemento di matrice se assegnata ad un elemento di matrice. RI numero di argomenti come appare nella lista CALL LINK. R2 indirizzo della stringa che deve essere assegnata. La stringa deve essere in RAM. Se l'argomento non é una variabile di stringa o un elemento di una matrice di stringa, é emessa una segnalazione di errore.

GET.NUMERIC PARAMETER - NUMREF

Formato: BLWP @NUMREF uguaglia NUMREF >6044

Questo programma ricava il valore di un parametro numerico.
R0 numero di elemento di matrice se l'argomento é una matrice numerica; altrimenti 0. RI numero di parametro come appare nella lista CALL LINK.
834A area FAC l'indirizzo di partenza di un valore di otto byte del parametro numerico, restituito dalla routine di gestione sistema.

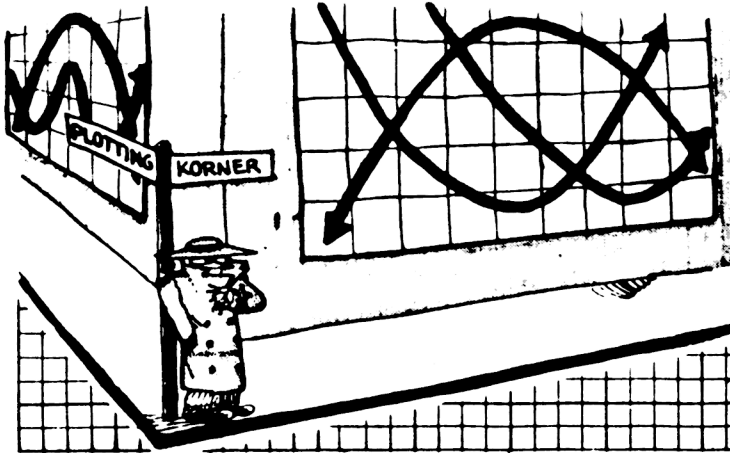
GET STRING PARAMETER - STRREF

Formato: BLWP @STRREF uguaglia STRREF a >604C

Questo programma ricava il valore di un parametro di stringa. Deve riservare spazio nella memoria RAM prima di chiamare questa routine, e il primo byte di questo BUFFER riservato deve contenere la lunghezza massima di buffer. Se la stringa non si inserisce, si hà una condizione di errore . R0 numero di elemento di matrice se l'argomento é una matrice di stringa; altrimenti zero. RI numero di parametro come appare nella lista degli argomenti di CALL LINK. R2 indirizzo del BUFFER che assegnate. Se la stringa si inserisce nel BUFFER vi viene trascritta seguendo il byte di lunghezza; il byte di lunghezza é modificato per riflettere la lunghezza attuale della stringa.

SEGNALAZIONE DI ERRORE - ERR

Formato: BLWP @ERR uguaglia ERR > 6050

Questo programma trasferisce il controllo alla routine di segnalazione errori nell'interprete TI BASIC . R0 codice di errore nel byte più significativo.
ATTENZIONE i codici di errore più piccoli di IO sono riservati al modulo MINIMEM. Quindi l'uso di questi codici nel programma può causare inprevedibili effetti secondari.

Plotting With The Home Computer

By Joseph G. DeVincentis, Jr.
P. O. Box 375
University of Dallas
Irving, TX 75061

The Assembly Language routines presented in this article will let Home Computer users draw axes, plot curves, and even draw objects in perspective. The software, consisting of plotting routines for the Mini Memory Cartridge, accesses the powerful graphics capabilities of the TI-99/4A through TI BASIC.

The routines supplied in this package require either of the following peripheral configurations:

1. Memory Expansion, cassette recorder, Mini-Memory, and assembled (object file) routines available on this issue's "99'er Magazine-On-Tape." [See page 69]
2. Memory Expansion, disk system, Mini-Memory, Editor/Assembler, and source files (hand entered from listings included with this article.)

```
*****************************
*          > BIT1 <         *
* PART ONE OF PLOTTING      *
*         ROUTINES          *
*****************************
* BY JOE DEVINCENTIS, JR.
* 99'ER VERSION 2.2.1ALMM
*
        TITL 'BIT MAP LINE'
*
        DEF  DRAW
        DEF  GCLEAR
        DEF  GRAPH
        DEF  LABEL
        DEF  MOVE
        DEF  SCALE
        DEF  XAXIS
        DEF  YAXIS
*
        REF  NUMREF
        REF  STRREF

A       EQU  5
ADR     EQU  0
ARG     EQU  >835C
B       EQU  6
BASE    EQU  >8343
CFI     EQU  >12B8
CHAR    EQU  5
CHRCNT  EQU  8
CNT     EQU  9
COLOR   EQU  >1000
COUNT   EQU  9
DELTAA  EQU  7
DELTAB  EQU  8
DRFLAG  EQU  4
ERR     EQU  10
ERRBA   EQU  >1600
ERRBS   EQU  >1700
ERRCOD  EQU  >8322
ERRNO   EQU  >1400
ERROR   EQU  >00CE
ERRSNM  EQU  >1500
ERRUN   EQU  >2500
FAC     EQU  >834A
FADD    EQU  >0D80
FCOM    EQU  >0D3A
FDIV    EQU  >0FF4
FLAG2   EQU  8
FLAG4   EQU  9
FMUL    EQU  >0E88
FSUB    EQU  >0D7C
GPLWS   EQU  >83E0
KEYCOD  EQU  >8375
MASK    EQU  6
MYPGT   EQU  >2800
ORIGIN  EQU  >7118
OVF     EQU  >8354
PCT     EQU  >2000
PGT     EQU  >0000
PNT     EQU  >1800
SCAN    EQU  >000E
SNT     EQU  >1800
STATUS  EQU  >837C
VDPRD   EQU  >8800
VDPRST  EQU  >8000
VDPWA   EQU  >8C02
VDPWD   EQU  >8C00
XP1     EQU  1
XP2     EQU  3
YP1     EQU  2
YP2     EQU  4
*
        AORG ORIGIN
DELTAX  DATA 0
DELTAY  DATA 0
SAVR11  DATA 0
STASAV  DATA 0
VAL255  DATA >4102,>3700,>0000
        DATA >0000
VAL191  DATA >4101,>5B00,>0000
        DATA >0000
VDPREG  DATA >8002,>81E0,>8206
        DATA >83FF,>8403,>8506
        DATA >8717
X       DATA 0
X1      DATA 0
Y       DATA 0
Y1      DATA 0
XSAVE   DATA 0
YSAVE   DATA 0
XDOT    DATA >4001,>0000,>0000
        DATA >0000
YDOT    DATA >4001,>0000,>0000
        DATA >0000
XMAX    DATA >4102,>3700,>0000
        DATA >0000
XMIN    DATA >0000,>0000,>0000
        DATA >0000
YMAX    DATA >4101,>5B00,>0000
        DATA >0000
YMIN    DATA >0000,>0000,>0000
        DATA >0000
*
BUFFER  BYTE >FF
        BSS  >FF
        EVEN
CONREG  BSS  >20
MREGS   BSS  >20
PREGS   BSS  >20
SREGS   BSS  >20
*
*
DRAW    MOV  @STATUS,@STASAV
        MOV  R11,@SAVR11
        BLWP @DRAW1
        LWPI GPLWS
        MOV  @SAVR11,R11
        MOV  @STASAV,@STATUS
        RT
*
*
GCLEAR  MOV  @STATUS,@STASAV
        MOV  R11,@SAVR11
        BLWP @CLEAR
        LWPI GPLWS
        MOV  @SAVR11,R11
        MOV  @STASAV,@STATUS
        RT
*
*
GRAPH   BLWP @VDPSET
        BLWP @GRPAH1
        LWPI GPLWS
SCANIT  BL   @SCAN
        MOVB @STATUS,R0
        JNE  SCANIT
        LI   R0,>0020
        MOVB @KEYCOD,R0
        CI   R0,'Q '
        JNE  SCANIT
        LI   R0,VDPRST
        BL   @VADR
        MOV  @2,R11
        B    *R11
*
*
LABEL   MOV  @STATUS,@STASAV
        MOV  R11,@SAVR11
        BLWP @LABEL1
        LWPI GPLWS
        MOV  @SAVR11,R11
        MOV  @STASAV,@STATUS
        RT
*
*
MOVE    MOV  @STATUS,@STASAV
        MOV  R11,@SAVR11
        BLWP @MOVE1
        LWPI GPLWS
        MOV  @SAVR11,R11
        MOV  @STASAV,@STATUS
        RT
*
*
SCALE   MOV  @STATUS,@STASAV
        MOV  R11,@SAVR11
        BLWP @SCALE1
        LWPI GPLWS
        MOV  @SAVR11,R11
        MOV  @STASAV,@STATUS
        RT
*
*
XAXIS   MOV  R11,@SAVR11
        MOV  @STATUS,@STASAV
        LI   R0,MREGS+16
        CLR  *R0+
        CLR  *R0
        BLWP @AXIS
        LWPI GPLWS
        MOV  @STASAV,@STATUS
        MOV  @SAVR11,R11
        RT
*
*
YAXIS   MOV  R11,@SAVR11
        MOV  @STATUS,@STASAV
        LI   R0,MREGS+16
        LI   R1,2
        MOV  R1,*R0+
        SLA  R1,1
        MOV  R1,*R0
        BLWP @AXIS
        LWPI GPLWS
        MOV  @STASAV,@STATUS
        MOV  @SAVR11,R11
        RT
*
*
AXIS    DATA MREGS,AXIS1
*
AXIS1   CLR  R0
        LI   R1,2
        BLWP @NUMREF
        LI   R2,FAC
        LI   R3,ARG
        BL   @TRDATA
        CLR  R0
        LI   R1,1
        BLWP @NUMREF
        LWPI GPLWS
        BL   @FCOM
        LWPI MREGS
        MOVB @STATUS,R0
        ANDI R0,>4000
        JGT  AXCON1
        LI   R0,ERRBA
        B    @ERRSYS
AXCON1  MOV  @X,@XSAVE
        MOV  @Y,@YSAVE
        LI   R2,ARG
        LI   R3,SREGS
        BL   @TRDATA
        MOV  FLAG2,R1
        SRA  R1,1
        AI   R1,1
        BLWP @CONVTR
        MOV  R0,@X(FLAG
        LI   R2,SREGS
        LI   R3,FAC
        BL   @TRDATA
        MOV  FLAG2,R1
        SRA  R1,1
        AI   R1,1
        BLWP @CONVTR
        MOV  R0,@X1(FLA
        S    @X(FLAG4),
        MOV  R0,@DELTAX
        CLR  R0
        LI   R1,3
        BLWP @NUMREF
        LI   R1,2
        MOV  FLAG4,FLAG
        JEQ  AXCON2
        SRA  R1,1
AXCON2  BLWP @CONVTR
        NEG  FLAG2
        NEG  FLAG4
        MOV  R0,@Y(FLAG
        MOV  R0,@Y1(FLA
        CLR  @DELTAY(FL
        BLWP @DRAW3
        MOV  @XSAVE,@X
        MOV  @YSAVE,@Y
        RTWP
*
*
        PAGE
```

```
*************************
*       > BIT2 <        *
* PART TWO OF PLOTTING  *
*       ROUTINES        *
*************************
* 99'ER VERSION 2.2.1ALMM
*
*LINE 0002
CLEAR  DATA MREGS,CLEAR1
CLEAR1 LI   R1,MYPGT
       LI   R2,>1800
CLCON1 CLR  *R1
       INCT R1
       DECT R2
       JNE  CLCON1
       CLR  @X
       CLR  @Y
       LI   R2,VAL255
       LI   R3,XMAX
       BL   @TRDATA
       LI   R2,VAL191
       LI   R3,YMAX
       BL   @TRDATA
       LI   R3,>4001
       MOV  R3,@XDOT
       MOV  R3,@YDOT
       CLR  @XMIN
       CLR  @YMIN
       LI   R1,3
       LI   R2,2
CLCON2 CLR  @XDOT(R2)
       CLR  @YDOT(R2)
       CLR  @XMIN(R2)
       CLR  @YMIN(R2)
       INCT R2
       DEC  R1
       JNE  CLCON2
       RTWP
*
*
CONVTR DATA CONREG,CONVT1
*
CONVT1 MOV  @2(R13),R5
       CI   R5,1
       JNE  CVCON1
       CLR  R5
       CLR  R6
       JMP  CVCON2
CVCON1 LI   R5,16
       LI   R6,8
CVCON2 LI   R2,FAC
       LI   R3,ARG
       BL   @TRDATA
       LI   R2,XMIN
       A    R5,R2
       LI   R3,FAC
       BL   @TRDATA
       LWPI GPLWS
       BL   @FSUB
       LWPI CONREG
       LI   R2,FAC
       LI   R3,ARG
       BL   @TRDATA
       LI   R2,XDOT
       A    R6,R2
       LI   R3,FAC
       BL   @TRDATA
       LWPI GPLWS
       BL   @FDIV
       BL   @CFI
       LWPI CONREG
       MOV  @FAC,*R13
       RTWP
*
*
DRAW1  DATA MREGS,DRAW2
DRAW3  DATA SREGS,DRAW4
*
DRAW2  CLR  R0
       LI   R1,1
       BLWP @NUMREF
       BLWP @CONVTR
       MOV  R0,@X1
       S    @X,R0
       MOV  R0,@DELTAX
       CLR  R0
       LI   R1,2
       BLWP @NUMREF
       BLWP @CONVTR
       MOV  R0,@Y1
       S    @Y,R0
       MOV  R0,@DELTAY
DRAW4  ABS  @DELTAX
       ABS  @DELTAY
       C    @DELTAY,@DELTAX
       JGT  DRCON1
       MOV  @DELTAX,DELTAA
       MOV  @DELTAY,DELTAB
       MOV  @X,A
       MOV  @Y,B
       CLR  DRFLAG
       JMP  DRCON2
DRCON1 MOV  @DELTAX,DELTAB
       MOV  @DELTAY,DELTAA
       MOV  @X,B
       MOV  @Y,A
       SETO DRFLAG
DRCON2 C    @X,@X1
       JLT  DRCON3
       SETO R2
       JMP  DRCON4
DRCON3 LI   R2,1
DRCON4 C    @Y,@Y1
       JLT  DRCON5
       SETO R3
       JMP  DRCON6
DRCON5 LI   R3,1
DRCON6 MOV  DELTAA,COUNT
       INC  COUNT
       MOV  DELTAB,R0
       SLA  R0,1
       MOV  DELTAA,R1
       S    R1,R0
       MOV  R0,ERR
       S    R1,R0
       MOV  DELTAB,R1
       SLA  R1,1
REDRW1 MOV  DRFLAG,DRFLAG
       JNE  REDRW2
       MOV  A,@X
       MOV  B,@Y
       JMP  REDRW3
REDRW2 MOV  A,@Y
       MOV  B,@X
REDRW3 BLWP @PLOT
       MOV  ERR,ERR
       JGT  CHNG
       A    R1,ERR
       JMP  INCR
CHNG   MOV  DRFLAG,DRFLAG
       JNE  REDRW4
       A    R3,B
       JMP  REDRW5
REDRW4 A    R2,B
REDRW5 A    R0,ERR
INCR   MOV  DRFLAG,DRFLAG
       JNE  REDRW6
       A    R2,A
       JMP  REDRW7
REDRW6 A    R3,A
REDRW7 DEC  COUNT
       JNE  REDRW1
       MOV  @X1,@X
       MOV  @Y1,@Y
       RTWP
*
*
ERRSYS MOV  R0,@ERRCOD
       LWPI GPLWS
       LI   R11,>000E
       MOV  *R11,R11
       B    @ERROR
*
*
GRPAH1 DATA MREGS,GRPAH2
*
GRPAH2 CLR  R0
       BL   @VADW
       LI   R1,>1800
       LI   R2,MYPGT
GRCON1 MOVB *R2+,@VDPWD
       DEC  R1
       JNE  GRCON1
       RTWP
*
*
LABEL1 DATA MREGS,LABEL2
*
LABEL2 CLR  R0
       LI   R1,1
       LI   R2,>FF00
       MOVB R2,@BUFFER
       LI   R2,BUFFER
       BLWP @STRREF
       MOV  @X,XP1
       CI   XP1,256
       JLT  LBCON1
       RTWP
LBCON1 CI   XP1,>8000
       JL   LBCON2
       RTWP
LBCON2 MOV  @Y,YP1
       CI   YP1,192
       JLT  LBCON3
       RTWP
LBCON3 CI   YP1,>8000
       JL   LBCON4
       RTWP
LBCON4 CLR  CHRCNT
       MOVB @BUFFER,CHRCNT
       SWPB CHRCNT
       NEG  YP1
       AI   YP1,191
       SRA  XP1,3
       SRA  YP1,3
       SLA  YP1,5
       A    XP1,YP1
       MOV  YP1,ADR
       A    CHRCNT,YP1
       CI   YP1,768
       JL   LBCON5
       LI   R6,768
       S    R6,YP1
       MOV  YP1,CHRCNT
LBCON5 LI   R7,BUFFER+1
       SLA  ADR,3
       AI   ADR,MYPGT
LLOOP1 CLR  CHAR
       MOVB *R7+,CHAR
       SWPB CHAR
       CI   CHAR,32
       JHE  LBCON6
       CLR  CHAR
       JMP  LBCON8
LBCON6 CI   CHAR,127
       JLE  LBCON7
       LI   CHAR,127
LBCON7 AI   CHAR,-32
       SLA  CHAR,3
LBCON8 AI   CHAR,CHRTBL
       LI   CNT,8
LLOOP2 SOC  *CHAR+,*ADR+
       CI   ADR,MYPGT+>1801
       JL   LBCON9
       RTWP
LBCON9 DECT CNT
       JNE  LLOOP2
       DEC  CHRCNT
       JNE  LLOOP1
       LI   R0,>FF00
       MOVB R0,@BUFFER
       RTWP
*
*
MOVE1  DATA MREGS,MOVE2
*
MOVE2  CLR  R0
       LI   R1,1
       BLWP @NUMREF
       BLWP @CONVTR
       MOV  R0,@X
       CLR  R0
       LI   R1,2
       BLWP @NUMREF
       BLWP @CONVTR
       MOV  R0,@Y
       RTWP
*
*
       PAGE
```

```
***************************
*        > BIT 3<         *
* PART THREE OF PLOTTING  *
*        ROUTINES         *
***************************
* 99'ER VERSION 2.2.1ALMM
*
*LINE 0002
*
PLOT    DATA PREGS,PLOT1
*
PLOT1   MOV  @X,XP1
        CI   XP1,256
        JLT  PLCON1
        RTWP
PLCON1  CI   XP1,>8000
        JL   PLCON2
        RTWP
PLCON2  MOV  @Y,YP1
        CI   YP1,192
        JLT  PLCON3
        RTWP
PLCON3  CI   YP1,>8000
        JL   PLCON4
        RTWP
PLCON4  MOV  XP1,XP2
        NEG  YP1
        AI   YP1,191
        MOV  YP1,YP2
        SRA  YP1,3
        SLA  YP1,5
        SRA  XP1,3
        MOV  YP1,ADR
        A    XP1,ADR
        SLA  ADR,3
        ANDI YP2,>0007
        A    YP2,ADR
        AI   ADR,MYPGT
        MOVB *ADR,CHAR
        ANDI XP2,>0007
        MOV  ADR,R4
        MOV  XP2,R0
        LI   MASK,>8000
        SRC  MASK,0
        MOV  R4,ADR
        SOC  MASK,CHAR
        MOVB CHAR,*ADR
        RTWP
*
*
SCALE1  DATA MREGS,SCALE2
*
SCALE2  CLR  R5
        CLR  R6
        BL   @NUMGET
        LI   R5,2
        LI   R6,16
        BL   @NUMGET
        CLR  R5
        CLR  R6
        BL   @NUMSUB
        LI   R5,8
        LI   R6,16
        BL   @NUMSUB
        CLR  R5
        BL   @NUMDIV
        LI   R5,8
        BL   @NUMDIV
        RTWP
*
NUMDIV  MOV  R11,R10
        LI   R2,VAL255
        A    R5,R2
        LI   R3,FAC
```

```
        BL   @TRDATA
        LI   R2,XDOT
        A    R5,R2
        LI   R3,ARG
        BL   @TRDATA
        LWPI GPLWS
        BL   @FDIV
        LWPI MREGS
        LI   R2,FAC
        LI   R3,XDOT
        A    R5,R3
        BL   @TRDATA
        B    *R10
*
NUMGET  MOV  R11,R10
        CLR  R0
        LI   R1,2
        A    R5,R1
        LI   R2,FAC
        LI   R3,XMAX
        A    R6,R3
        BLWP @NUMREF
        BL   @TRDATA
        DEC  R1
        LI   R2,FAC
        LI   R3,XMIN
        A    R6,R3
        BLWP @NUMREF
        BL   @TRDATA
        LI   R2,XMAX
        A    R6,R2
        LI   R3,ARG
        BL   @TRDATA
        LWPI GPLWS
        BL   @FCOM
        LWPI MREGS
        MOVB @STATUS,R0
        ANDI R0,>4000
        JGT  SCCON1
        LI   R0,ERRBA
        B    @ERRSYS
SCCON1  B    *R10
*
NUMSUB  MOV  R11,R10
        LI   R2,XMIN
        A    R6,R2
        LI   R3,FAC
        BL   @TRDATA
        LI   R2,XMAX
        A    R6,R2
        LI   R3,ARG
        BL   @TRDATA
        LWPI GPLWS
        BL   @FSUB
        LWPI MREGS
```

```
        LI   R9,>0100
        CB   R9,@OVF
        JNE  SCCON2
        LI   R0,ERRNO
        B    @ERRSYS
SCCON2  CLR  R7
        CLR  R8
        MOVB @FAC,R7
        SWPB @FAC
        MOVB @FAC,R8
        ABS  R7
        ABS  R8
        MOVB R8,@FAC
        SWPB @FAC
        MOVB R7,@FAC
        LI   R2,FAC
        LI   R3,XDOT
        A    R5,R3
        BL   @TRDATA
        B    *R10
*
*
TRDATA  MOV  *R2+,*R3+
        MOV  *R2+,*R3+
        MOV  *R2+,*R3+
        MOV  *R2,*R3
        RT
*
*
VADW    ORI  R0,>4000
VADR    SWPB R0
        MOVB R0,@VDPWA
        SWPB R0
        MOVB R0,@VDPWA
        RT
*
*
VDPSET  DATA MREGS,VDP1
*
VDP1    LI   R2,VDPREG
        LI   R1,7
VDCON1  MOV  *R2+,R0
        BL   @VADR
        DEC  R1
        JNE  VDCON1
        LI   R0,SNT
        LI   R1,>D000
        BL   @VADW
        MOVB R1,@VDPWD
        LI   R0,PCT
        BL   @VADW
        LI   R1,COLOR
        LI   R2,>1800
VDCON2  MOVB R1,@VDPWD
        DEC  R2
```

```
        JNE  VDCON2
        LI   R0,PNT
        BL   @VADW
        LI   R3,3
VDCON3  CLR  R1
        LI   R2,256
VDCON4  MOVB R1,@VDPWD
        AI   R1,>0100
        DEC  R2
        JNE  VDCON4
        DEC  R3
        JNE  VDCON3
        RTWP
*
*
*
        PAGE
```

```
***************************
*          > BIT4 <       *
* PART FOUR OF PLOTTING   *
*        ROUTINES         *
***************************
* 99'ER VERSION 2.2.1ALMM
*
*LINE 0002
*
CHRTBL DATA >0000,>0000,>0000,>0000  blank
       DATA >0010,>1010,>1010,>0010  !
       DATA >0028,>2828,>0000,>0000  "
       DATA >0028,>287C,>287C,>2828  #
       DATA >0038,>5450,>3814,>5438  $
       DATA >0060,>6408,>1020,>4C0C  %
       DATA >0020,>5050,>2054,>4834  &
       DATA >0008,>0810,>0000,>0000  '
       DATA >0008,>1020,>2020,>1008  (
       DATA >0020,>1008,>0808,>1020  )
       DATA >0000,>2810,>7C10,>2800  *
       DATA >0000,>1010,>7C10,>1000  +
       DATA >0000,>0000,>0030,>1020  ,
       DATA >0000,>0000,>7C00,>0000  -
       DATA >0000,>0000,>0000,>3030  .
       DATA >0000,>0408,>1020,>4000  /
       DATA >0038,>4444,>4444,>4438  0
       DATA >0010,>3010,>1010,>1038  1
       DATA >0038,>4404,>0810,>207C  2
       DATA >0038,>4404,>1804,>4438  3
       DATA >0008,>1828,>487C,>0808  4
       DATA >007C,>4078,>0404,>4438  5
       DATA >0018,>2040,>7844,>4438  6
       DATA >007C,>0408,>1020,>2020  7
       DATA >0038,>4444,>3844,>4438  8
       DATA >0038,>4444,>3C04,>0830  9
       DATA >0000,>3030,>0030,>3000  :
       DATA >0000,>3030,>0030,>1020  ;
```

```
       DATA >0008,>1020,>4020,>1008   <
       DATA >0000,>007C,>007C,>0000   =
       DATA >0020,>1008,>0408,>1020   >
       DATA >0038,>4404,>0810,>0010   ?
       DATA >0038,>445C,>545C,>4038   @
       DATA >0038,>4444,>7C44,>4444   A
       DATA >0078,>2424,>3824,>2478   B
       DATA >0038,>4440,>4040,>4438   C
       DATA >0078,>2424,>2424,>2478   D
       DATA >007C,>4040,>7840,>407C   E
       DATA >007C,>4040,>7840,>4040   F
       DATA >003C,>4040,>5C44,>4438   G
       DATA >0044,>4444,>7C44,>4444   H
       DATA >0038,>1010,>1010,>1038   I
       DATA >0004,>0404,>0404,>4438   J
       DATA >0044,>4850,>6050,>4844   K
       DATA >0040,>4040,>4040,>407C   L
       DATA >0044,>6C54,>5444,>4444   M
       DATA >0044,>6464,>544C,>4C44   N
       DATA >007C,>4444,>4444,>447C   O
       DATA >0078,>4444,>7840,>4040   P
       DATA >0038,>4444,>4454,>4834   Q
       DATA >0078,>4444,>7850,>4844   R
       DATA >0038,>4440,>3804,>4438   S
       DATA >007C,>1010,>1010,>1010   T
       DATA >0044,>4444,>4444,>4438   U
       DATA >0044,>4444,>2828,>1010   V
       DATA >0044,>4444,>5454,>5428   W
       DATA >0044,>4428,>1028,>4444   X
       DATA >0044,>4428,>1010,>1010   Y
       DATA >007C,>0408,>1020,>407C   Z
       DATA >0038,>2020,>2020,>2038   [
       DATA >0000,>4020,>1008,>0400   \
       DATA >0038,>0808,>0808,>0838   ]
       DATA >0000,>1028,>4400,>0000   ^
```

```
       DATA >0000,>0000,>0000,>7C00  _
       DATA >0000,>2010,>0800,>0000  `
       DATA >0000,>0038,>447C,>4444  a
       DATA >0000,>0078,>2438,>2478  b
       DATA >0000,>003C,>4040,>403C  c
       DATA >0000,>0078,>2424,>2478  d
       DATA >0000,>007C,>4078,>407C  e
       DATA >0000,>007C,>4078,>4040  f
       DATA >0000,>003C,>405C,>4438  g
       DATA >0000,>0044,>447C,>4444  h
       DATA >0000,>0038,>1010,>1038  i
       DATA >0000,>0008,>0808,>4830  j
       DATA >0000,>0024,>2830,>2824  k
       DATA >0000,>0040,>4040,>407C  l
       DATA >0000,>0044,>6C54,>4444  m
       DATA >0000,>0044,>6454,>4C44  n
       DATA >0000,>007C,>4444,>447C  o
       DATA >0000,>0078,>4478,>4040  p
       DATA >0000,>0038,>4454,>4834  q
       DATA >0000,>0078,>4478,>4844  r
       DATA >0000,>003C,>4038,>0478  s
       DATA >0000,>007C,>1010,>1010  t
       DATA >0000,>0044,>4444,>4438  u
       DATA >0000,>0044,>4428,>2810  v
       DATA >0000,>0044,>4454,>5428  w
       DATA >0000,>0044,>2810,>2844  x
       DATA >0000,>0044,>2810,>1010  y
       DATA >0000,>007C,>0810,>207C  z
       DATA >0018,>2020,>4020,>2018  {
       DATA >0010,>1010,>0010,>1010  |
       DATA >0030,>0808,>0408,>0830  }
       DATA >0000,>2054,>0800,>0000  ~
       DATA >0000,>0000,>0000,>0000  del
*
*
       END
```

```
***************************
*       > SOURCE <        *
*                         *
* PART FIVE OF PLOTTING   *
*        ROUTINES         *
***************************
*
*99'ER VERSION 2.2.1ALMM
*
       COPY "DSK1.BIT1"
       COPY "DSK1.BIT2"
       COPY "DSK1.BIT3"
       COPY "DSK1.BIT4"
       END
```